NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

SOFTWARE REUSABILITY:
A DECISION TREE MODEL

by

William D. Randall, Jr.

June 1988

Thesis Advisor: Gordon H. Bradley

Approved for public release; distribution unlimited.

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b OFFICE SYMBOL (*If applicable*) 33	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (*City, State, and ZIP Code*) Monterey, California 93943-5000	7b. ADDRESS (*City, State, and ZIP Code*) Monterey, California 93943-5000
8a. NAME OF FUNDING / SPONSORING ORGANIZATION 8b OFFICE SYMBOL (*If applicable*)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (*City, State, and ZIP Code*)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11. TITLE (*Include Security Classification*)

SOFTWARE REUSABILITY: A DECISION TREE MODEL

12. PERSONAL AUTHOR(S)
Randall, William D., Jr.

13a. TYPE OF REPORT	13b TIME COVERED	14. DATE OF REPORT (*Year, Month, Day*)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	1988, June	73

16. SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U. S. Government.

17. COSATI CODES			18 SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*)
FIELD	GROUP	SUB-GROUP	Reusable software, decision trees, decision modeling software engineering

19. ABSTRACT (*Continue on reverse if necessary and identify by block number*)

There are numerous claims in the software engineering literature that reusable software will solve many of the problems extant in the software industry, but there are few articles examining the economic factors inherent in the reusability issues. This thesis proposes a decision tree as a model of the reuse decision and suggests application for its use.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Gordon G. Bradley	22b TELEPHONE (*Include Area Code*) 408 646 2359 22c. OFFICE SYMBOL 52Bz

Approved for public release; distribution is unlimited.

Software Reusability: A Decision Tree Model

by

William D. Randall, Jr.
Lieutenant Commander, United States Navy
BCHE, Auburn University, 1975

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ENGINEERING SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1988

ABSTRACT

There are numerous claims in the software engineering literature that reusable software will solve many of the problems extant in the software industry, but there are few articles examining the economic factors inherent in the reusability issues. This thesis proposes a decision tree as a model of the reuse decision and suggests applications for its use.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION AND BACKGROUND

A. WHY REUSABLE SOFTWARE?

In his thesis, written in March 1984, LCDR William C. Johnson described the software crisis as a "situation within the computer industry in which production and maintenance of computer systems is 'bottlenecked' by the software components systems." [Ref. 1] In June 1984, T. C. Jones stated, "The average productivity rates of industrial and commercial software builders have been in the vicinity of 2,000 to 4,000 lines of source code per programmer year, since the mid-1960's." [Ref. 2] If the productivity rates since 1984 have remained fairly constant, and there is no indication to the contrary, the "crisis" still exists. In fact, it must be getting worse.

Maintenance is another major concern of software engineers. The costs of maintaining software are increasing; in 1985, one writer states:

> The percentage of time that programmers spend on maintenance has been steadily increasing for the last several years. Computerworld's annual DP budget survey [CW, Dec. 31, 1984/Jan. 7, 1985] found that the percentage of an average programmer's time spent on maintenance now stands at about 55%, compared with 45% last year. There are no indications that this figure will go anywhere but up. [Ref. 3]

There have been many articles advocating software reusability as a remedy to the software crisis, but there appear to be few articles that indicate that these software

concerns are abating, or that software engineers are turning to reusable software in order to ease these concerns.

B. REUSABLE SOFTWARE

In some sense, software engineers do reuse software. Software engineers know the algorithms and code for certain problems that they have faced often in the course of their careers. They have developed their own tool kits which they carry with them from project to project, from job to job. When similar problems arise, they rely on this software tool kits to solve the problem or complete the project at hand. There are other examples of reuse that commonly occur.

Romberg and Thomas [Ref. 4] suggest that reusable software may encompass anything from the use of subroutines in Basic programs to the continued reuse of a single application program. This indicates that the meaning of the phrase is ambiguous and also depends on the context in which it is presented. It is certainly true that one reuses software every time he uses a commercial word processor or a commercial spread sheet. However, within the context of this thesis the phrase "reusable software" will refer to software modules retained in a library that are available to the software engineers for their use in software design projects. Hence, the word "module" will refer to atomic program components that form the fundamental building blocks of complete programs or software systems. A module may be a

single algorithm, or a collection of algorithms which collectively achieve a single end.

The module need not and, perhaps, should not be coded. Software is limited by its environment. It is very closely tied to the compiler version, the operating system and the hardware for which it was written. A software module can thus be quite limited in its reusability. To avoid this limitation, the module could consist of the requirements and specifications to which the software is written, the algorithms used to achieve the requirements and specifications, the interfaces of the software module, the testing and test plans and the documentation. In this thesis, the word "module" will refer to just that.

If several people are working on a project and if two or more require the use of the same abstract data type, it is wasteful for these people to each write their individual data type. It makes much more sense for the abstract data type to be written and then made available for all concerned with the project. A complete description of the data type (which could include the algorithms used to construct it, the types of data it can hold, the types of operations that can be performed on it, the requirements for which it was written and the testing to which it has been subjected) could be retained in a library of software modules. Subsequently, when the data type was again needed, the

module could be retrieved from the library. This is reusable software within the context of this thesis.

C. CLAIMS OF REUSABILITY OF SOFTWARE

Reusable software has been cited as a means of increasing software productivity, improving software reliability and reducing software costs. "Reusability reduces software development costs, speeds up the development process, and reduces testing needs." [Ref. 5] Further support of reliance on reusable software is found in an article by Romberg and Thomas:

> Production of one piece of code to do the work that would otherwise require many coding efforts creates obvious savings of time and effort in specification, design, construction and testing. These savings are minimally offset by the need to meet the interface requirements of a large set of environments in which the code will be used. The net effect is substantially reduced development time and cost. The savings are compounded as the resources freed are applied to additional development. [Ref. 6]

Despite these ringing endorsements for the use of reusable software in the software industry, there is little evidence to indicate that reusability is being embraced by software engineers. In his 1984 book, David King [Ref. 7] describes a project library with no mention of coded or uncoded modules that may be reused by the program engineers and in 1986, William Wong [Ref. 8] of the Institute of Computer Sciences and Technology (ICST) of the National Bureau of Standards stated that analysts and programmers

generally have to build code from scratch, and that few organizations have an organized body of knowledge of software assets that would allow an analyst or programmer to find and reuse modules. If reusability is worthy enough to have as large a share of the current software engineering literature devoted to it as does, why is reusability not receiving the attention of software engineers in industry?

This thesis proposes a model that may explain why software reusability is not in widespread use in the industry, and how, when it is appropriate, it might be encouraged.

II. RATIONAL DECISIONS/ECONOMIC ANALYSIS

A. RATIONAL DECISIONS

If one reads the literature, he or she is led to believe that any software engineer should be anxious to reuse software at any opportunity that presented itself. Software that has already been written, that is pre-designed to meet design specifications, that has been tested again and again, and, finally, that is free of design and production costs, would seem to be a ripe plum, ready for the picking. The fact is, few software engineers reuse software modules. When software engineers are faced with a software design project they examine the specifications then fit their experiences and knowledge to the problem. The software engineer is likely to determine how the current project is like or unlike projects in the past and apply the knowledge gained in the past to the current project.

This type of behavior is not unlike renting a car. Even though the car may be a make that is different from the make that a driver is accustomed to driving, the driver can still make decisions about driving the rental by relying on knowledge about the other make. The ignition switch will be on the dash, or on the steering column. The ignition must be "on" in order to unlock the gear lever and the steering wheel. The light switch is on the dash near the steering

column and the turn signal is the small lever on the left of the steering column.

Similarly, when a software engineer is faced with a problem in software design his solution is based on what has worked in similar situations in the past. What data types or algorithms worked best in this situation the last time? When last faced with this question did a linked list or simple array work better? Is a trapazoidal rule or a Simpson's rule approximation better in this case?

This is rational behavior and software engineers are rational people. They will make design decisions based on sound judgments. Economically speaking, the soundest judgment is the decision which results in a combination of low risk, low cost and high return.

B. ECONOMIC ANALYSIS

In order to understand the software engineer's reluctance to rely on reusable software in accomplishing his assignment, one should undertake an economic analysis of his decision, as he surely does. The economic analysis is based on choices. Each of these choices is characterized by risk, expected value and cost. The software engineer makes the choices he does because he feels that they are economically sound choices.

A particular decision may mean fewer lines of code, and therefore, less time and money spent on coding this segment

of the project. A second decision may be made because one alternative is more likely to yield the expected results. Or, a third decision may be made because it will lead to increased profits for the firm or a particular department within the firm.

These are economic considerations; they appear to be absent from much of the current literature in the field of reusability. Yet, if reusable software is ever to become a reality in the software industry these considerations will require much more attention than they are getting at the present time.

III. MODELLING THE REUSE DECISION

A. RATIONAL DECISIONS MADE EVERY DAY

Every day, people make rational decisions, either consciously or unconsciously evaluating each alternative available to them and the consequences of pursuing these alternatives. Based on this evaluation, and the perceived answers to questions about consequences, a decision is reached. The decision is to pursue the alternative which returns the best combination of return, risk and cost.

The difficulty is in determining which alternative leads to the optimal combination of return, risk and cost. One method in overcoming this difficulty is decision analysis, through the use of the decision tree.

B. DECISION TREES

Decision trees are one tool used by management in the analysis of decisions. This discussion of decision trees is based on discussions of decision trees in texts by Ackoff and Sasieni [Ref. 9] and Markland and Sweigart [Ref. 10].

A decision tree is a graphical representation of a decision-making process that provides the decision maker with a stage-by-stage account of a decision. A decision tree consists of branches, representing events which might occur, and nodes, representing the points at which alternatives occur. There are two types of nodes; decision

nodes, represented by circles, at which one has the opportunity to elect one alternative over others, and chance nodes, represented by squares, at which nature controls the outcome. A node may have two or more branches. Figure 1 shows a simple decision tree with two branches.

C. STAIRS OR ELEVATOR?

At this point, an example may prove to be useful. Consider a man who has a meeting on the fourth floor of a certain building. On entering the building, he has the choice of taking the elevator to the fourth floor, or he may take the stairs. (Figure 2 shows the decision tree that represents this decision.)

In the decision tree, events are shown in chronological order, from left to right. Node A is a decision node, indicating that this person has the choice of taking the stairs, branch S, or of taking the elevator, branch E. At the end of each of these branches, are chance nodes, marked B and C. From each of these chance nodes, emanate two branches marked "Success" and "Failure", the two possible states of nature resulting from the decision maker's earlier choice of taking the stairs or of taking the elevator.

The decision maker makes his decision by evaluating and comparing the expected values of his alternatives. First, he determines the payoff for success in each case. In this example, the payoff is the same for both the stairs and the

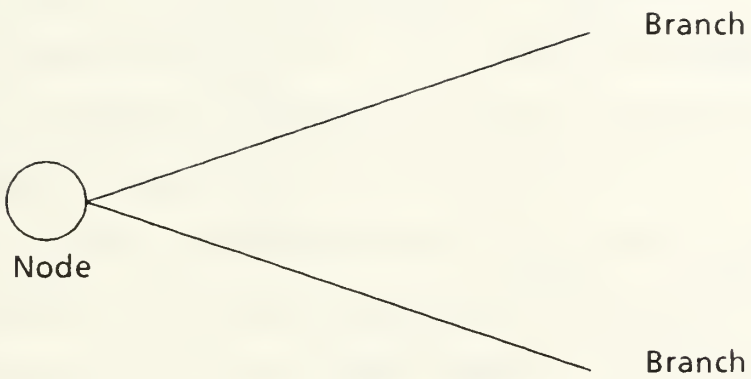

Figure 1. Simple Decision Tree

elevator. That is, whether he arrives at the fourth floor by stairs, or by elevator, he will still be in time for his meeting. Because the payoffs are the same for each case, a relative value of 1.00 can be assigned to each of the Success branches in Figure 2. If, for whatever reason, the man does not arrive at the fourth floor (Failure), the payoff is 0.00, again, as shown in Figure 2.

The decision maker must then determine the probability of success or failure at each chance node. (Any probability must be between 0.00 and 1.00.) If, in the experience of the decision maker, the elevator works reliably, he may assign the event that he arrives at his destination a probability of 0.99. (It could never by 1.00, as there is always the possibility of malfunction.) If, on the other hand, the elevator is under repair, or it is the experience of the decision maker that the elevator is rarely in operation, he will assign his arrival by elevator a very low probability, one very nearly 0.00. It is assumed, for this example, that the elevator is in good repair and that the decision maker assigns his arrival by elevator a probability of 0.99. He will assign his arrival at his destination by stairs a probability of 1.00, because a power failure will not cause the stairs to fail, as it would the elevator.

Since this event can only have one of two outcomes, the probability of success and the probability of failure must sum to 1.00. Therefore, whatever probability is assigned to

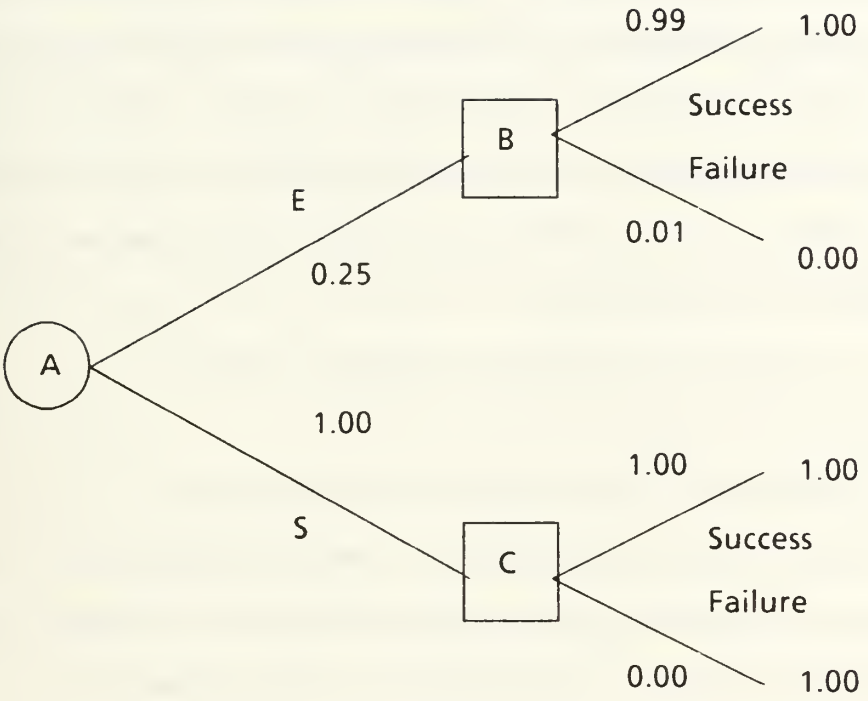

Figure 2. Decision Tree for Stairs vs. Elevator Decision

the Success branch, the Failure branch must be assigned a value of 1.00 less the probability of success. At node B, for example, the probability of failure is 1.00 - 0.99, or 0.01. All probabilities are shown on Figure 2.

Next, the decision maker calculates the expected value at each of the chance nodes. This is done by taking the product of the probability and the payoff of each state of nature at one chance node, then summing them. This sum is assigned as the expected value of that node. Other chance nodes are evaluated similarly. In this illustration, the expected value of node B is

$$0.99 \times 1.00 + 0.01 \times 0.00 = 0.99$$

and the expected value of node C is

$$1.00 \times 1.00 + 0.00 \times 0.00 = 1.00.$$

There is a cost associated with each alternative. For instance, it may take fifteen seconds for the elevator to reach the fourth floor, but walking, it requires sixty seconds to reach the fourth floor. Thus, the rational decision maker may, consciously or otherwise, assign the elevator alternative a relative cost of 0.25, but the stairs alternative a relative cost of 1.00, since it "costs" four times as much time to take the stairs as it "costs" to take the elevator. These costs, shown in Figure 2, are subtracted from the expected values just calculated to yield revised expected values of 0.74 for the elevator (0.99 - 0.25) and 0.00 for the stairs (1.00 - 1.00).

The decision maker now compares the expected values of each of the alternatives and makes the rational decision to take the elevator. However, it is possible that different values for some of the variables will result in a different decision. For example, if the decision maker places great value on exercise, he may judge the payoff from taking the stairs to be twice the payoff of taking the elevator, the savings in time to the contrary, assuming that he can still arrive in time for his appointment. A payoff of 2.00 for arriving at his destination by stairs changes the expected value of node C to 1.00 ([1.00 x 2.00 + 0.00 x 0.00] - 1.00 = 1.00). It is now rational to take the stairs to the fourth floor.

Note that the expected values were calculated from the end of the decision tree, to the beginning. This is similar to the thought processes in making a decision. One might think to himself, "Either way, I get to the fourth floor. If I take the stairs, I'm sure that I will get there, but it's going to take much longer. On the other hand, if I take the elevator, I'm almost as sure of getting to the fourth floor and I will get there more quickly. I'll take the elevator."

Other variables can be changed with no effect on the decision. If the payoffs are the same as originally stated, but the elevator is not as reliable as in the first illustration, the probability of success of arriving by

elevator may be degraded to 0.50. This changes the expected value of node B to (0.50 x 1.00 + 0.50 x 0.00) - 0.25 = 0.25. Even though the probability of success in taking the elevator is nearly half of its value in the original illustration, the expected value from taking the elevator (0.25) is still greater than the expected value of taking the stairs (0.00) and the rational decision does not change.

D. EXAMPLE EXTENDED TO SOFTWARE ENGINEERING

Although the preceding is a rather simple example, it illustrates the use of decision trees in everyday decisions. Further, it can be extended easily to the environment of software engineering.

Having arrived at the fourth floor, this rational decision maker (a software engineer) is given a software project to complete. The project is to write a software module for subsequent inclusion in a larger project. However, he is now faced with a series of choices instead of a single choice. These choices can also be represented by a decision tree, albeit a more complex decision tree, as shown in Figure 3. The decision tree contains five decision nodes, A, B, D, E and G. These represent the points at which the software engineer has the opportunity to make a decision about the use of reusable software in the current project. The decision tree also contains two chance nodes, C and F. These are the points at which the software

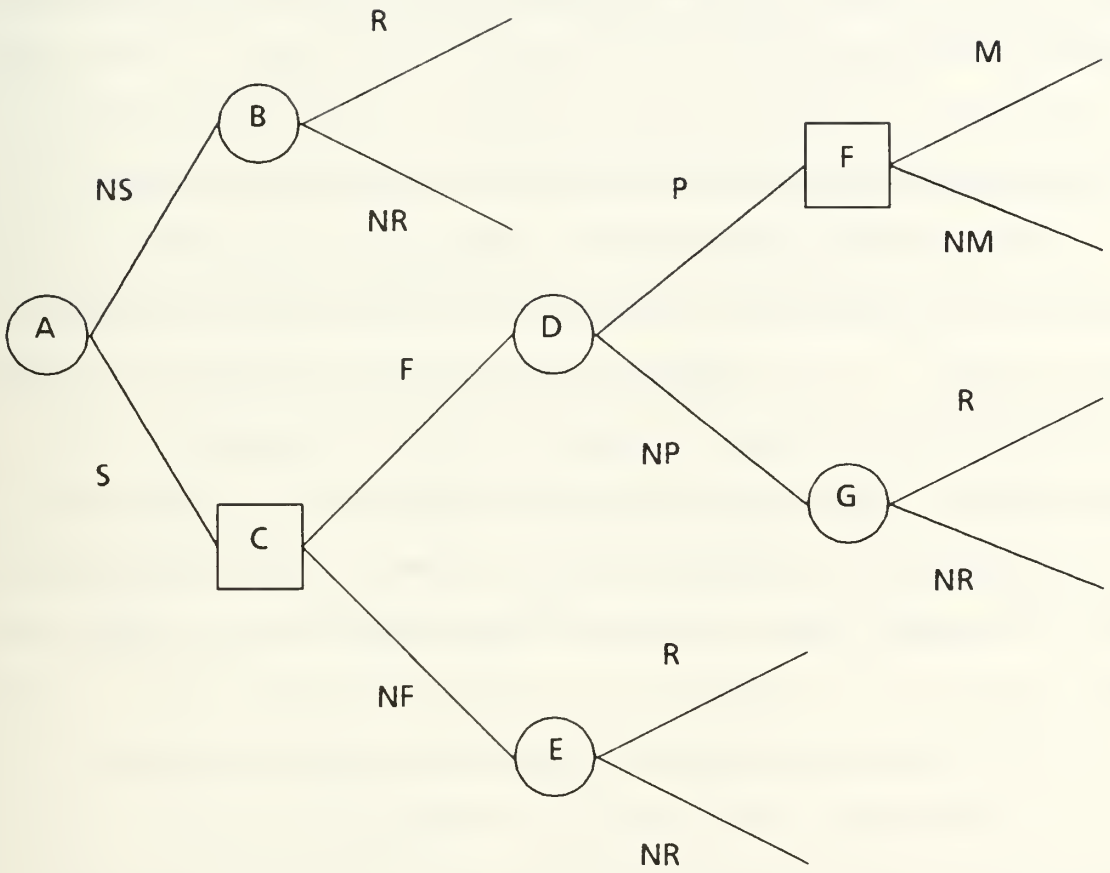

Figure 3. Reuse Decision Tree

engineer has no control over the state of nature which follows. In this decision tree, there are eight branches which do not terminate in a node. These indicate that a final state has been reached. Two of these branches emanate from chance node F and two emanate from each of the decision nodes B, E and F.

E. DESCRIPTION OF THE REUSE DECISION THROUGH A MODEL

The first node, labelled A, represents the decision of the software engineer to create his own software modules from scratch, or to conduct a search of a software library for applicable modules, already written and tested. The upper branch represents the alternative that no search is conducted and is labelled NS. The lower branch represents the alternative that a search is conducted and is labelled S.

If the software engineer elects to take the branch labelled NS, he arrives at another decision node, B. At this juncture, he may elect to create a software package that is suitable to his purpose, but without consideration that it be used in similar applications in the future (i.e., reusable), the branch labelled NR. This will be referred to as conventionally written software. Alternatively, the software engineer, may elect to create a software package that meets the strict standards of the reusable software library, represented by the branch labelled R. Having done

so, when faced with similar assignments in the future, this or other software engineers, may reuse the software created now.

The software engineer may decide to conduct a search of the library of reusable software, represented by the branch labelled S. This option results in arriving at chance node C. Here, the software engineer has no election. The two branches stemming from node C are mutually exclusive, chance events. The search reveals a software module that the software engineer may be able to use, the F branch, or the search reveals nothing, the NF branch.

If the search reveals nothing, the software engineer is at decision node E, which is identical to node B. However, should the search reveal a software module, the decision tree takes the software engineer to decision node D. At node D, the software engineer must decide if he is going to pursue the reusable software option, branch P, or if he is going to abandon his efforts to reuse and create the software from scratch, the NP branch. The NP branch takes the software engineer to decision node G, again, identical to decision node B.

However, if the software engineer decides to pursue the reusable software option further and takes branch P, he arrives at chance node F. This node represents the point that the module revealed by the search may or may not require some modification before the software engineer can

use the module in his project, the states of nature M and NM, respectively. (The alternatives of modifying or not modifying may not be entirely up to chance. The software engineer may have some discretion in this matter. This issue will be discussed at a later point.)

This decision tree is proposed as the model of the decision making process of the software engineer engaged in a large-scale software project. Ensuing discussion will center around examples using arbitrarily chosen data, the problems of data collection as it pertains to this model and application of the model to the software engineering environment.

IV. CALCULATIONS/SENSITIVITY ANALYSIS

A. REFINED EXAMPLE

In order to facilitate the model and the following example, the following variables are defined:

* P_m -- the probability of having to modify a software module.
* P_{nm} -- the probability of not having to modify a software module; $P_{nm} = 1.00 - P_m$.
* P_f -- the probability of finding the software module that meets the specifications in a software library.
* P_{nf} -- the probability of not finding the software module that meets the specifications in a software library; $P_{nf} = 1.00 - P_f$.
* C_{ab} -- the cost of proceeding from node A to node B; that is, the overhead involved in starting out writing the software from scratch.
* C_{ac} -- the cost of proceeding from node A to node C; that is, the overhead involved in searching a software library for a reusable modules.
* C_{br} -- the cost of writing a reusable module, having arrived at node B.
* C_{bnr} -- the cost of writing a non-reusable module, having arrived at node B.
* C_{df} -- the cost of examining modules revealed by the

search, to determine which module best fits the requirements.

* C_{er} -- the cost of writing a reusable module, having arrived at node E.

* C_{enr} -- the cost of writing a non-reusable module, having arrived at node E.

* C_{gr} -- the cost of writing a reusable module, having arrived at node G.

* C_{gnr} -- the cost of writing a non-reusable module, having arrived at node G.

* C_{fm} -- the overhead involved in using a software module that requires modification.

* C_{fnm} -- the overhead involved in using a software module that requires no modification.

* V_{fm} -- the payoff realized for using a reusable module that requires modification.

* V_{fnm} -- the payoff realized for using a reusable module that requires no modification.

* V_{bnr} -- the payoff realized for producing a software module that is not reusable, after decision node B.

* V_{br} -- the payoff realized for producing a software module that is reusable, after decision node B.

* V_{enr} -- the payoff realized for producing a software module that is not reusable, after decision node E.

* V_{er} -- the payoff realized for producing a software module that is reusable, after decision node E.

* V_{gnr} -- the payoff realized for producing a software module that is not reusable, after decision node G.

* V_{gr} -- the payoff realized for producing a software module that is reusable, after decision node G.

The expected value of each node will be denoted by the letter designation of that node. In other words, D is, for example, the expected value at decision node D. These variables are shown in Figure 4, on the branches on which they are encountered. They will be used to illustrate the facility and application of the model in the following discussions.

B. CALCULATIONS WITH ARBITRARILY CHOSEN DATA

Figure 5 is an amended copy of Figure 4, showing the values assigned for this illustration. Each of the branches which do not terminate in a node show the payoffs associated with its particular state. Both branches emanating from chance node F show a payoff of $40,000. Branches emanating from nodes B, E and G show payoffs of $50,000. These data are arbitrarily chosen, and are used only to illustrate the analytical capabilities of the model.

Assume that a realization of $50,000 is forecast for the completion of this phase of the project, unless the module is a reusable module found in the library as a result of search. In this case, the realization may only be $40,000, as the module may be less than the exact specifications.

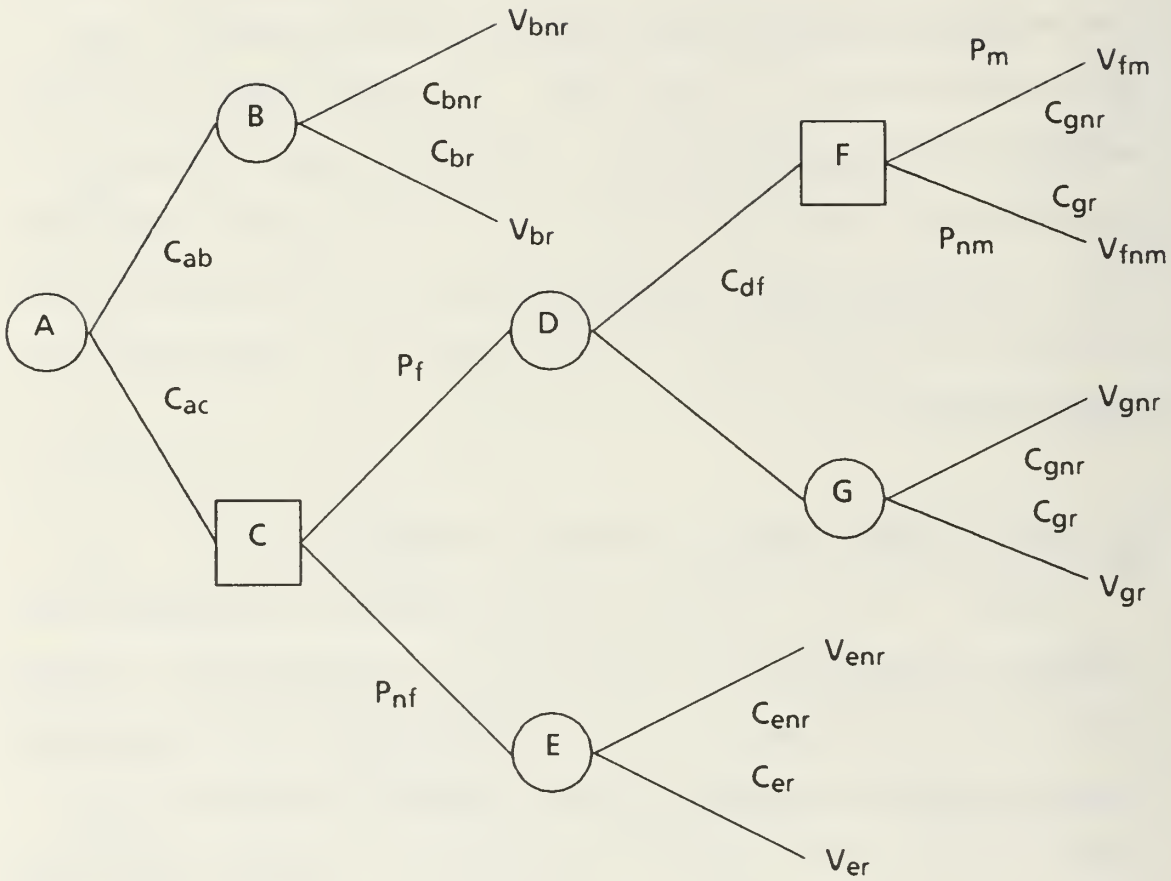

Figure 4. Decision Tree Showing Variables

Further, assume that any branch representing a chance event has an equal chance of occurrence ($P_m = P_{nm} = P_f = P_{nf} = 0.50$).

It will be assumed, for this illustration, that the cost of writing a software module in the conventional manner is $25,000 and that the cost of writing a software module that is reusable is $30,000, a 20% increase over writing a non-reusable module. Therefore, $C_{bnr} = C_{enr} = C_{gnr} = \$25,000$ and $C_{br} = C_{er} = C_{gr} = \$30,000$.

The cost of modifying a software module is, in this illustration, assumed to be $20,000 ($C_{fm} = \$20,000$). At this point, it is easy to jump to the conclusion that there is no overhead if one is fortunate enough to find a reusable module that exactly fits the specifications. This is probably not the case, as it will no doubt have some interface with other modules in the overall project. Based on this reasoning, in this illustration, $C_{fnm} = \$5,000$. With these data, the decision of the software engineer is evaluated as described in the following paragraphs.

The expected value of node F is evaluated, as follows:

$$F = P_{fm}(V_{fm} - C_{fm}) + P_{fnm}(V_{fnm} - C_{fnm}).$$

Substituting the assumed values into the above formula yields the following results:

$$F = 0.50(40,000 - 20,000) + 0.50(40,000 - 5,000)$$

$$F = \$27,500.$$

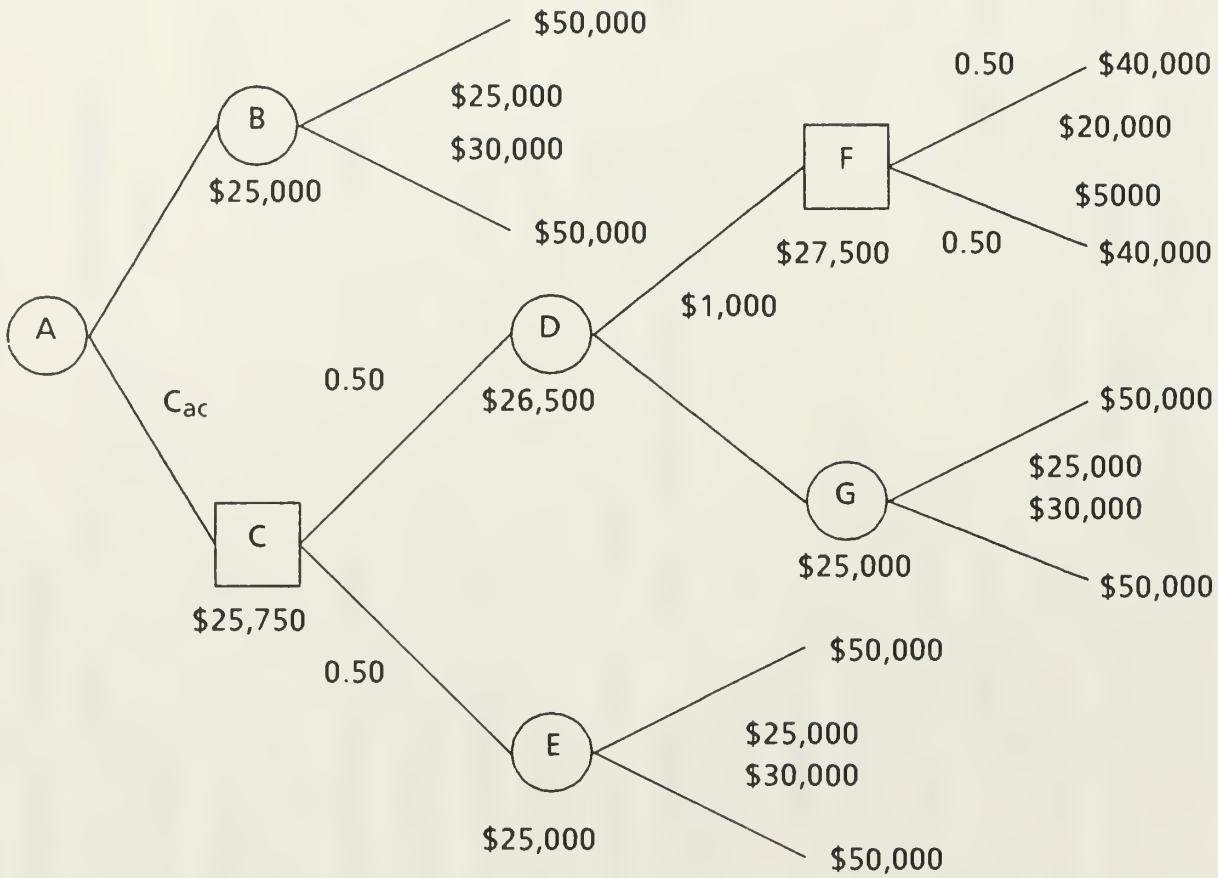

Figure 5. Reuse Decision Tree Showing Assigned Values

The expected value of node G is computed by finding the difference between the payoff and the cost of each branch and selecting the greater value. For example:

$$V_{gnr} - C_{gnr} = 50,000 - 25,000 = \$25,000$$

and

$$V_{gr} - C_{gr} = 50,000 - 30,000 = \$20,000.$$

Thus the expected value of G is \$25,000. (Likewise, the expected values of nodes B and E are \$25,000.)

The expected value of node D is the greater of the values $(F - C_{df})$ and G. F was previously calculated as \$27,500, but must be decreased by the cost of examining the retrieved modules for the best fit. Assuming C_{df} = \$1000, $(F - C_{df})$ = \$26,500. G was shown to be \$25,000. Therefore, the expected value at node D is \$26,500.

The expected value of node C is calculated as follows:

$$C = (P_f \times D) + (P_{nf} \times E).$$

Substituting for the variables yields the following results:

$$C = (0.50 \times 26,500) + (0.50 \times 25,000)$$

$$C = \$25,750.$$

The expected value of node A is the greater of the expected values of nodes B and C, less any costs associated with the two branches. If the cost of the library search is \$1,500, the expected value of node A is \$25,000, but can be achieved only by taking the path marked NS. The path marked S yields only \$24,250 (25,750 - 1500). Based on this

illustration, the software engineer's decision is to write the required software module from scratch.

C. THE PROBABILITY OF MODIFICATION, P_m

P_m is the probability of having to modify a software module that has been identified as at least marginally meeting the specifications of the software project on which the software engineer is working. In the example cited above, P_m was given a value of 0.50. This indicates that half of the software modules called upon for reuse will require some modification in order to fit the bill. This can still be economically advantageous to the software engineer, as long as the cost of the modification is not too great.

Initially, this probability is likely to be much higher than 0.50. As the library is filled with more and better reusable software modules, the probability that its modules will require modification will decrease. Additionally, P_m can be decreased by more strictly enforcing acceptability requirements to the library. Thus, P_m might be viewed as a measure of the quality of the software library.

D. THE PROBABILITY OF FINDING A MODULE, P_f

P_f is the probability that the library will contain a reusable software module that can be used by the software engineer. In the example, P_f was set at 0.50, which

indicates that for every 100 searches of the library, 50 will be fruitful. This obviously depends on the quantity of the library contents. As the library is filled with more and more reusable software modules, the probability that a reusable software module is found increases. P_f can be viewed as a measure of the quantity of the library.

E. SENSITIVITY OF THE MODEL

A model, if it is a good analytical tool, must be able to demonstrate the effects of changes of one or more of the parameters on other parameters. In the case of the decision tree as a model of the software engineer's decision process, the model should, for example, be able to demonstrate the effect a change in the probability of modification, P_m, on the decision to search or write from scratch; or the effect of a change in the payoff resulting from writing a reusable module, rather than writing a module in a conventional manner. Does the proposed model accomplish this? Subjectively, one would surmise that if the probability of having to modify a module were quite large, say 0.95, it might have some effect on the software engineer's decision concerning a library search. Will the model support this instinctive feeling?

A change of the value of P_m to 0.95 will change the expected values of the decision and chance nodes as shown in Figure 6 and as follows. The expected value at node F is

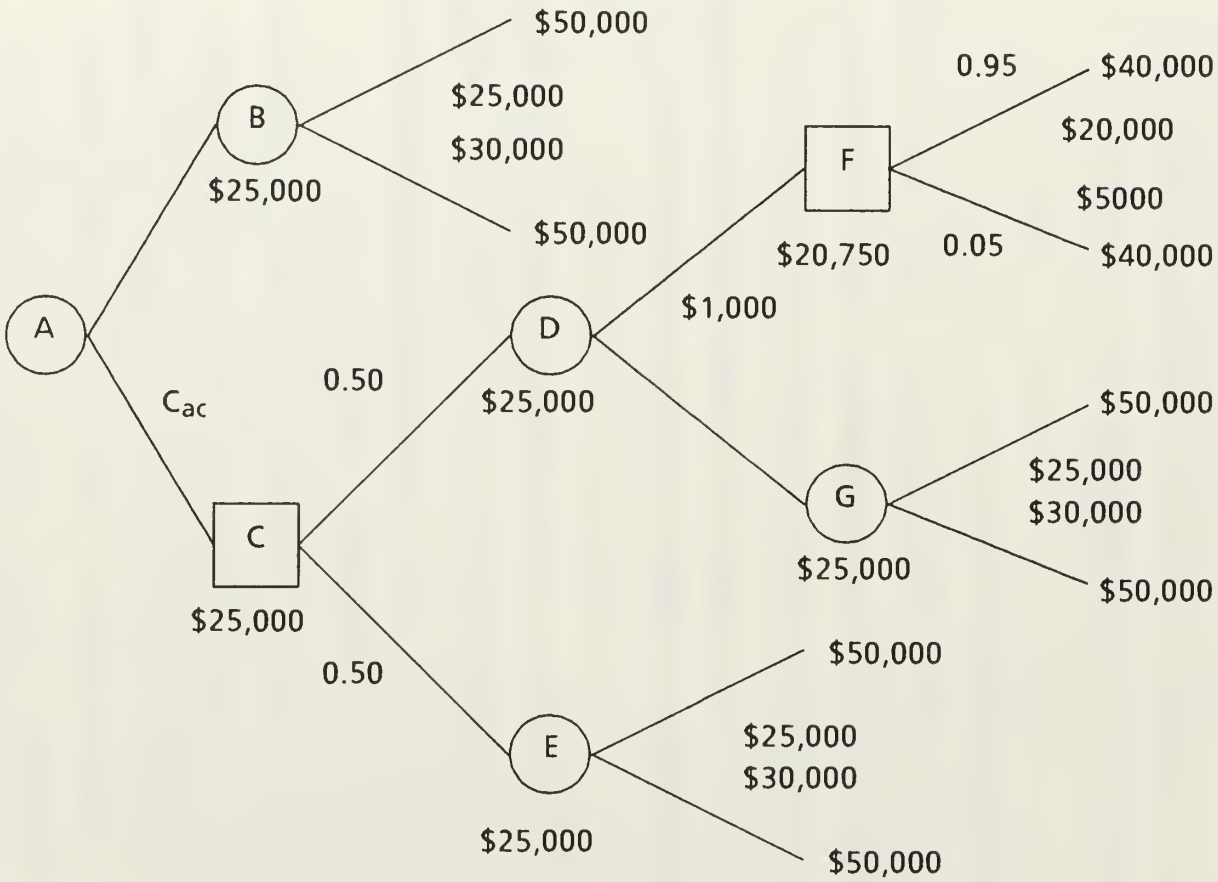

Figure 6. $P_m = 0.95$

reduced to the amount of $20,750. With an expected value at F of that amount, the expected value at node D changes from $26,500 to $25,000, since $25,000 is the larger of the expected values at F and G. Still assuming that P_f is 0.50, the expected value at node C is $25,000, vice $25,750 in the original example. This means that whatever the overhead involved in conducting the search for a reusable software module, the economically advantageous route is to choose to write the software from scratch, as one instinctively believes.

Another variable that might be changed is the payoff realized when a reusable module is used. In the example, the payoff in this case was $40,000, predicated on the observation that the module may be slightly less than that which was specified. But, if the software engineer is going to modify the module (branch M), the final product may be very much closer to that which was specified and therefore worth more than $40,000. What if the payoff for a modified module is $45,000, as shown in Figure 7?

With the payoff for branch M valued at $45,000 (and P_m again set to 0.50), the expected value of F is $30,000, driving the expected value of D to $29,000. This drives the expected value at C to $27,000, which means that a search of the software library is economically advantageous so long as the cost of conducting the search is less than $2,000.

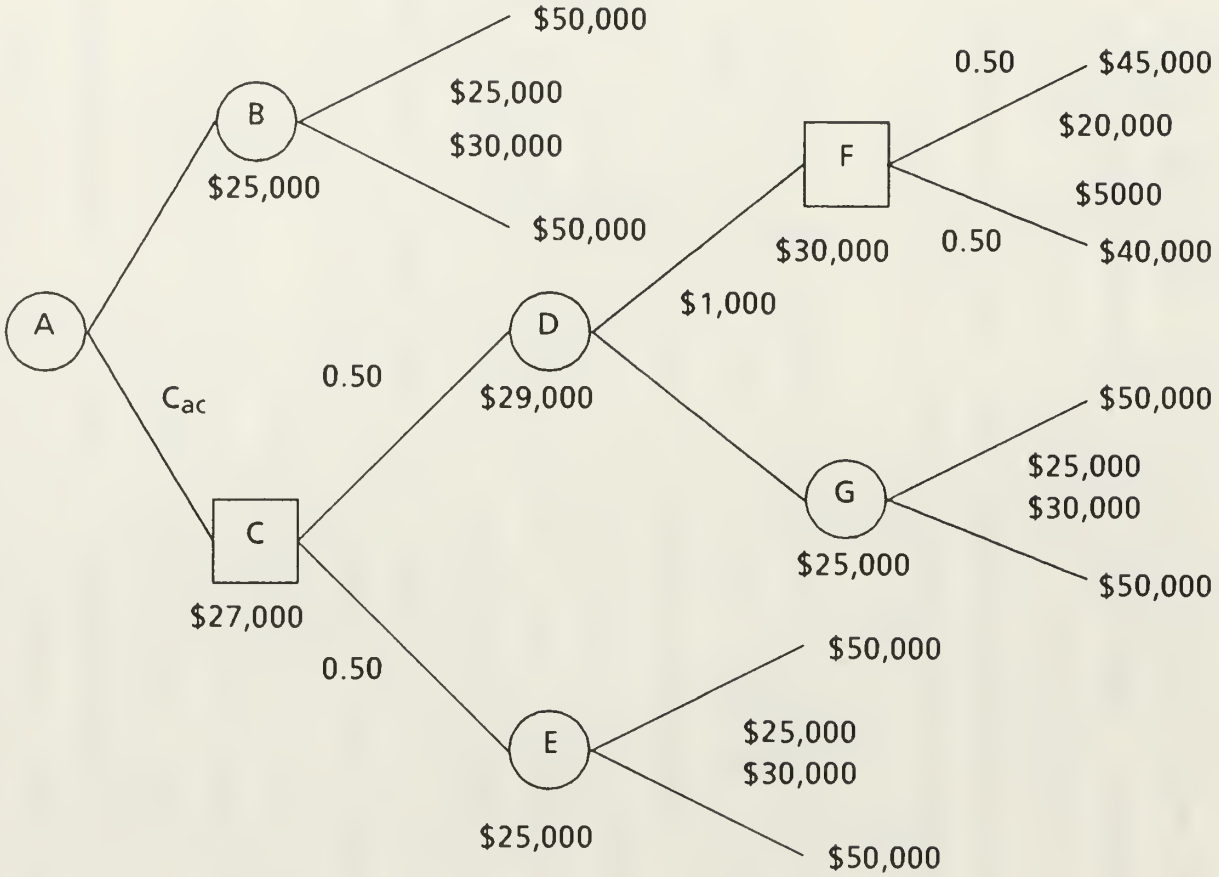

Figure 7. V_{fm} = $45,000

F. COMPUTER PROGRAMS OF THE DECISION TREE MODEL

Two computer programs were written to demonstrate the analytic capabilities of the model. They are included as an appendix to this thesis. The programs are written in the Pascal programming language. The variables are the costs, expected values, payoffs and probabilities, as explained in the model and illustrated in the example. The first program is written using the data found in the example. However, rather than assuming a value of P_m, the probability that a module will require modification, the program is written so that it will find the value of P_m at which it is no longer economically rational to conduct a library search for reusable software modules. This is as if the software engineer, or his supervisor, were to have a reasonably good estimate of the value of P_f, the probability of finding an applicable reusable module in the library and was interested in determining what value P_m would have to be in order for a library search to be rational.

This first program is written assuming a value of P_f of 0.50, as in the example. A "while" loop, incrementing P_m from 0.00 to 1.00, in steps of 0.01, is written. At each increment of P_m the program determines the expected value of node B (Rb), the expected value of node C (Rc) and the branch selected by the decision maker (S or NS for Search and No Search, respectively). The program prints this information for each increment. With this particular set of

data, the rational decision is to conduct the search of the library until the value of P_m is 0.47.

The second program is written to establish a relationship between P_f and P_m. To establish this relationship, all variables from the model, except P_f and P_m, are assigned values. A "while" loop, incrementing P_m from 0.00 to 1.00, in steps of 0.01, is written. At each increment of P_m, P_f is incremented from 0.00 to 1.00, in steps of 0.01. For each increment of P_f, the program determines whether the rational (economically advantageous) decision is to conduct a search of the library, or to write the software module from scratch. For each value of P_m, the program prints the values of P_m, the minimum value of P_f for which a search is economically sound and the return expected as a result of the decision.

The computations carried out by either of these programs are easily done analytically. However, the short computer programs cited in this thesis will allow the decision maker to quickly examine the effects of varied sets of data. For example, the first computer program assumed a value of P_f of 0.50 and found that at a value of P_m of 0.47, the rational decision was to write the software from scratch. Changing the value of P_f to 0.75 and running the program, the decision maker can easily ascertain that P_m can be as high as 0.52 and search still be economically feasible.

V. DATA COLLECTION

A. DIFFICULTY IN OBTAINING DATA

Up to this point in the thesis, all the data has been arbitrarily chosen to illustrate the model of the software decision. This was simply because accurate data applicable to this topic is extremely difficult to obtain. If, as Boehm [Ref. 11] says, the annual cost per software professional is $44,650 and the average software professional has an average output of 2,000 lines per year, then a rough per-line cost for software is $22.32. Using this figure, the modules in the example would have to be approximately 1100 lines each.

The difficulty is in how these costs are determined. What is a software professional, an analyst, a programmer or an "average" of the two. To determine costs, does one include more than salary, such as utilities and office overhead? Likely, the answers to these questions are different for different groups and would thus have to be answered individually.

The assignment of values to the probabilities is an even more difficult task. How does one establish the probability of finding a usable module in a library. The best way is through history. A certain number of successes were achieved in a known number of searches. This percentage can be used as the value of P_f, but is, until the number of searches is large, subject to a wide margin of error.

The assignment of a value to the probability that a module will have to be modified (P_m), is even more difficult. This difficulty is exacerbated by the fact that the node at F, although presented as a chance node, can not be placed neatly into the category of chance or decision node. The issue of whether a module is modified is not entirely up to chance. Two different software engineers may have differing opinions about the applicability of a particular software module, with or without modification. Hence, this node shows attributes of both chance and decision nodes. It is only because of a lack of precise representation of the cognitive aspects of this issue that node F is presented as a chance node.

VI. APPLICATION OF MODEL

A. INITIALIZING THE LIBRARY

In many cases, there may not exist a library of reusable software modules. If it does exist, the library is sparsely populated. This forces the software engineer, according to the model, to write the software from scratch. The literature suggests that writing reusable software is more difficult and more expensive than writing software in a conventional manner, meaning that no reusable modules are being written to place in such a library. Therefore, the task facing the software industry is one of overcoming a powerful inertia in software engineering.

One scheme to encourage the expansion of the library of software might be to offer bonuses for reusable software modules over conventionally written software. How might the model help the software industry? Suppose the manager of a team of software engineers was considering such a bonus. How much of a bonus can realistically be offered? How will the bonus effect the rest of the operation?

In an organization just getting started in reusability, with a very small or non-existent library, P_f, the probability the needed software module is present, is going to be extremely small. The results of the second computer program representing the model show that in the ideal situation, i.e., perfectly reusable modules ($P_m = 0.00$), the

probability of finding a reusable module in the library must be 0.12 in order for the search to be an economically feasible alternative. Obviously, a P_f of 0.12 is not possible if the library is empty.

The model shows, that in this situation, the software engineer will elect to write the module from scratch; indeed, there is no other choice. As a decision maker, the software engineer will follow the NS branch, to decision node B. At node B, the software engineer will have to make a decision whether to write the software as a reusable module or conventionally. Again, the model shows that the software engineer will decide to write the software conventionally, as this is the alternative which provides the greatest return. As long as the rational decision is to write software in the conventional manner, no reusable modules will be created to add to the library. As long as no reusable modules are being added to the library, the rational decision is to write software in the conventional manner. The project supervisor, examining the decision making model, can predict this cycle and may be provided a clue to the solution.

The project supervisor can see that the library must be provided a base of reusable modules. He can also see that the rational decision is to write software conventionally. It simply costs too much to write reusable software. For the software engineer to make the decision to write reusable

software, it must become economically feasible. For writing reusable software to become economically feasible, either the cost of writing the module must be reduced, or the payoff of a reusable module must be higher.

Reducing the cost of writing a software module is a difficult task, if possible at all. However, it may be possible for the software firm to have the department responsible for the management of the library to underwrite a portion of each reusable module that it finds acceptable. In his 1984 article, T. C. Jones [Ref. 12] states that some concerns have gone so far as to establish the reusable library as an overhead item or as a cost center. If this scheme removes all cost of writing from the particular software project (but only if the module is written for reuse), any decision maker arriving at decision node B should make the decision to write a reusable software module.

However, the model shows that with this scheme, the decision maker will never decide to conduct a search of the library. It is now more profitable to write the reusable module, regardless of what is in the library. This will result in a very well stocked library, but one which suffers greatly from duplication. One way to guard against this is for the software firm to stipulate that the library cost center will underwrite the cost of writing a reusable module, once a library search has been conducted and the

library contained no module to meet the required specifications. This requires the decision maker to follow the decision tree to decision node E, as a minimum. This scheme drives the software engineers to stock the library, but without the risk of placing redundant modules in the library.

Does this cost center have to underwrite the entire cost of the reusable software module? An examination of Figure 8 will help reveal the answer. For an empty library, P_f is 0.0; P_{nf} is 1.0. This means that the expected value of chance node C is due solely to the contribution of the NF branch and, because P_{nf} is 1.0, the contribution of the NF branch is the expected value of decision node E. Knowing that C_{ac} is \$1000.00, the software engineer, or his manager, can see that the expected value of node C less the cost of conducting a search has to be greater than the expected value of node B, \$25,000. For this to occur, in these circumstances, the condition $V_{er} - C_{er} > \$26,000$ must be true. In the limiting case, $V_{er} - C_{er} = \$26,000$ and for the case of $V_{er} = \$50,000$, $C_{er} = \$24,000$. Therefore, if the library cost center underwrites \$6,001 of the \$30,000 to write a reusable module, the rational decision maker will follow the decision tree to decision node E, even when he knows that P_f is 0.0. Further, once at decision node E, he will follow branch R, writing a reusable software module, that can be placed in the software library for future use.

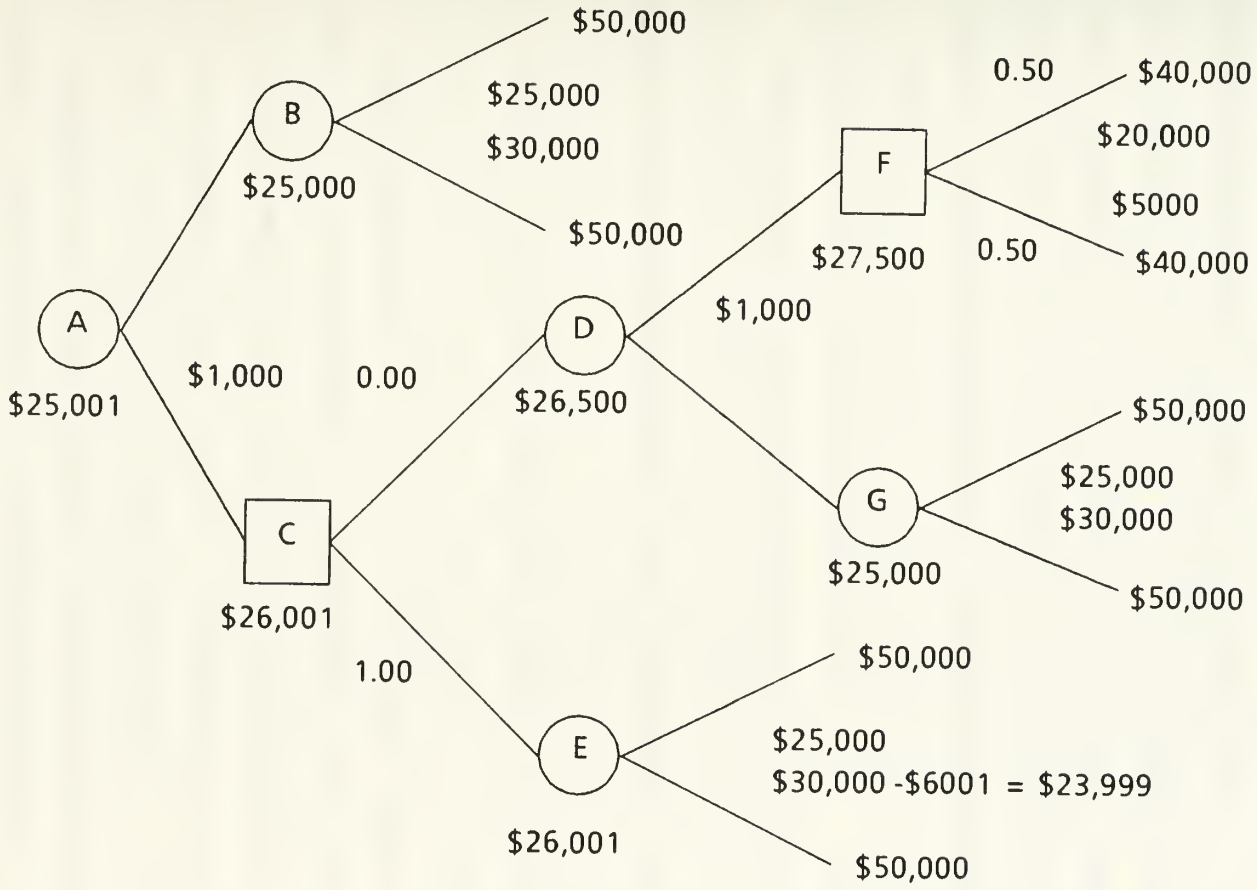

Figure 8. Underwriting the Costs

In the previous paragraph, the library's cost center underwrote a portion of the costs of writing the reusable module. This has results identical to the results had another cost center provided a bonus of the amount underwritten. The important aspect is that net payoff of this branch be greater than the net payoff of the NR branch, not the manner in which the advantage was achieved.

How long should the cost center continue to underwrite a portion of the cost of writing reusable modules? As the library grows under this expansion policy, P_f increases as P_{nf} decreases. Thus, the contribution of the NF branch to the expected value of chance node C diminishes, as the contribution of the F branch to the expected value of node C increases. At some point, the relative contributions should be such that, even with the cost center withdrawing its support, the economically sound decision is to conduct the library search.

To determine this point, consider the example shown in Figure 5. Each of the probabilities was set to 0.50, for purposes of illustration. This resulted in an expected value of $25,750 at node C. If the cost of conducting the library search is less than $750, the rational decision is to proceed with the search. However, that one conducts a search does not guarantee that a reusable module is found. The decision maker may find himself at decision node E,

having to decide between writing a reusable module or writing a conventionally structured software module.

At node E, the decision maker will again follow the branch which provides the greater return. If following branch NR results in a return of $25,000 and following branch R results in a return of $20,000, the rational software engineer will follow branch NR. The model shows that the cost center should continue to underwrite some portion of the cost of writing a reusable software module.

B. CAN THE LIBRARY BE TOO LARGE?

If the library is too well stocked, the software engineer may be overwhelmed by the number of modules revealed by the search. A cost is associated with the branch from node D to node F. This cost, C_{df}, reflects the investment in time and money incurred by the software engineering team in sorting a large number of modules to find the best of the lot. In the example (Figure 5), with the expected value of node G valued at $25,000, the rational software engineer will choose the branch from node D to node F so long as C_{df} is less than $2,500 ($27,500 - $25,000). (Of course, this decision may simply manifest itself as the software engineer's throwing up his hands in despair at such an overwhelming number of modules from which to choose, for which the model makes no allowance.)

The question of how many modules are too many is another question for which the model has no explicit answer. However, if the cost of examining a number of modules for the best candidate can be determined, the number of modules for which the cost surpasses an acceptable limit may be ascertainable.

C. A NATIONAL LIBRARY OF SOFTWARE

As an example of a software library that may be too large, consider the following. There are advocates of a national library of software which would hold every piece of software ever written or commissioned by the federal government. Such an endeavor is ambitious and appealing. What does the proposed decision tree model reveal about this suggestion?

First, with a library consisting of thousands (millions?) of programs, one can assume a large value of P_f, a value very nearly 1.00. It is very likely that anything one wants would be in such a large library. However, this might also be the problem. Such a library is bound to have a great deal of redundancy in it. This redundancy manifests itself by resulting in an overwhelming amount of software that is applicable to the project at hand being revealed by the search.

The increase in the amount of software which must be considered by the software engineer results in an increase

in the value of C_{df} as the engineer examines the software to determine which best meets his needs. (See Figure 4.) It is possible that the value of C_{df} will become so high that no matter what the expected value of node F is, the value of C_{df} will reduce the expected value of node F to a value below that of the expected value of node G. Therefore, the rational decision will be to write the software from scratch. The model has shown that in such a library guards against excessive redundancy must be in place and has also provided a method of determining how much redundancy is excessive.

Excessive redundancy is not the only force effecting the value of C_{df}. The form of the contents of the library can also lead to an increase in the value of C_{df}. If the contents of the library are complete programs, or are modules with incomplete or badly written specifications, C_{df} may again be so high that the software engineer will write his software from scratch. Here, the model shows that the national software library must place strict rules on the contents of the library.

It was shown earlier that in order to start a library of reusable software modules, the library was going to have to underwrite a portion of the cost of writing the reusable software modules, or to offer a bonus for reusable software modules, over conventionally written software. In a national library containing software written under the

auspices of the government, the situation is different. Producing software for the government, the software engineer would conduct a search of this national library. (It could even be made a stipulation of the contract.) If the library search reveals nothing, what will induce the software engineer write a reusable software module? This is a case in which a bonus is applicable, vice underwriting a portion of the costs. (In effect, the government is already underwriting the production costs, anyway.) The model can be used by the contract managers or by the library managers to determine the size of the bonus necessary.

D. GRAPHICAL ANALYSIS OF P_f vs P_m

The second computer program representing the decision making model was run three times; once, with the values used in the example and shown in Figure 5 and twice with revised values for V. This was done to observe the effect on the relationship of P_m and P_f. From the data output by the program on each of these three runs, a graph was prepared and is included as Figure 9.

Curve 1 is the case which was used as the example. The curve shows that as the probability of having to modify increases (the quality of the reusable modules decreases), the probability of finding the applicable reusable module must increase (the quantity of the reusable modules must increase). In this case, P_m must be higher than 0.50 before

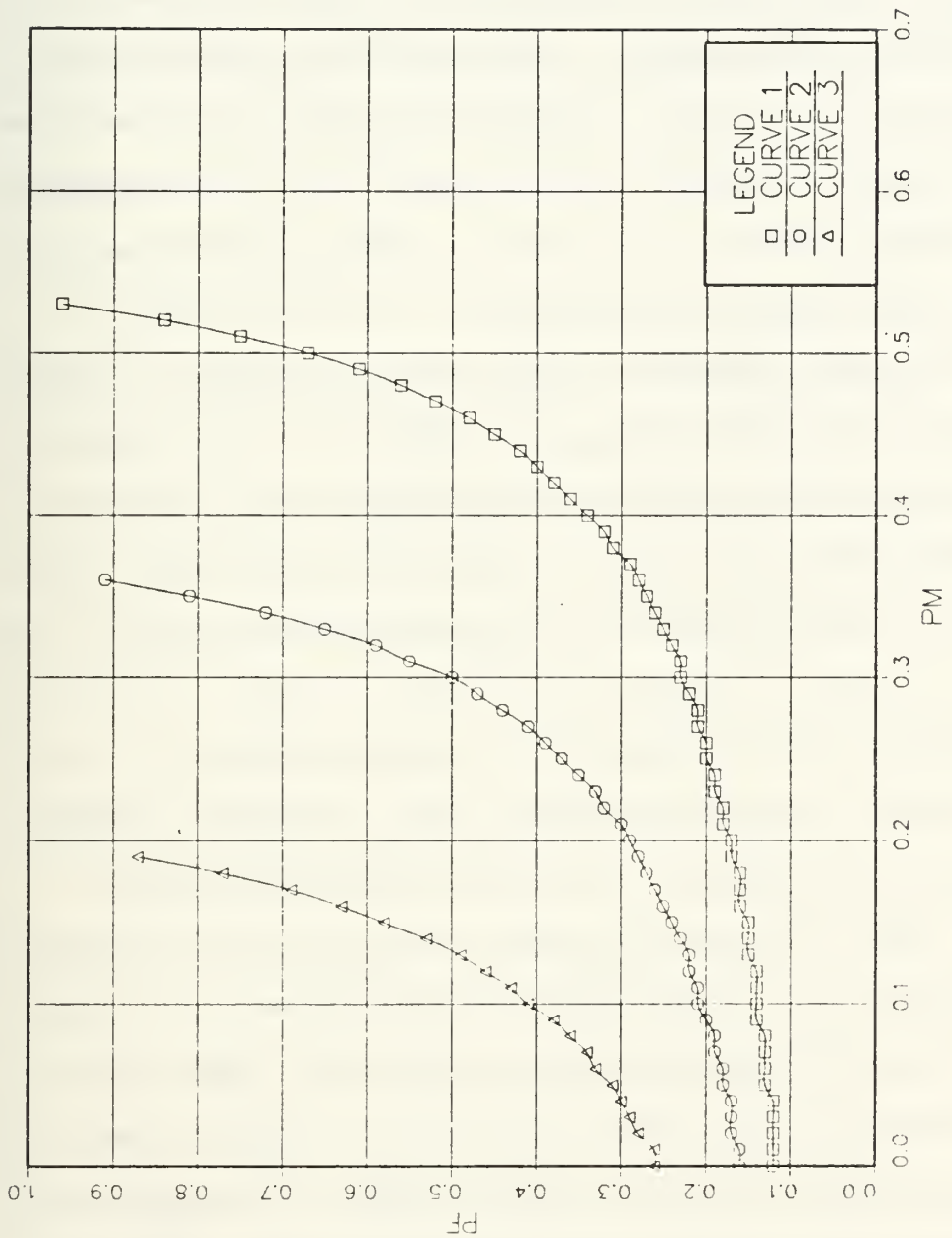

Figure 9. P_f vs P_m

the chance of finding a reusable module has to be nearly 1.00.

Curves 2 and 3 represent the increase in value of V of 15% and 20%, respectfully. These curves show the same general shape as does curve 1, but increasing toward 1.00 with smaller values of P_m. From this graph, it can be seen that with greater increases in the payoff a stricter quality control mechanism must be in place in order to keep the values of P_f in a reasonable range.

With only minor changes to the computer program, it will provide data in the form of P_m as a function of P_f. This could be advantageous if the data relevant to P_f is more readily accessible than is data relevant to P_m. This is an illustration of the flexibility of the proposed model.

E. WHERE SHOULD DATA COLLECTION BE CONCENTRATED?

Great expense can be incurred in the collection of data to be used in forecasting the feasibility reusability in software. There are many variables involved in this model and if attempts were made to collect data for each of these variables, the result may be some wasted time, effort and money at the least. Are there, then, some variables that do not have significant effects on the outcome of the software engineer's decision to use or not to use reusable software modules?

The model exhibits a sensitivity to changes in the costs of writing software, or to changes in the payoffs associated with the various branches of the decision tree. This indicates that the firm of software engineers considering the possibility of initiating a software library would do well to take a long, hard look at their cost of writing software and at the payoffs likely to result from the completion of projects.

It does not seem to be a sensible use of one's time to gather a great deal of data on the probabilities involved in the model. If the firm is only now embarking on a path to reusability, the probability of finding the right module is zero. The library is empty. It is not until the library has begun to be established that any data about P_f is even available. Then it becomes desirable to track P_f as the value of P_f can determine the amount of cost that must be underwritten by the library cost center.

Similarly, it does not seem to be a sensible use of one's time to gather a great deal of data on P_m. Again, the data is not available until the library is established and in use. Once the library is established P_m should be tracked in order to provide those concerned with a measure of the quality of the reusable modules in the library. If P_m is high, it indicates that the modules are frequently in need of modification, that is, the modules being written to

stock the library are in need of stricter quality assurance measures.

F. WHEN DOES LIBRARY PAY FOR ITSELF?

It has been estimated that the start-up costs for a reusable software library will fall into a range of $50,000 to upwards of $250,000 [Ref. 13]. This is a sizable investment and software engineers will have to know if this investment can be recovered. As this is an important aspect to the reusability issue, the model proposed in this thesis should assist in ascertaining this break-even point. Does it?

To answer this question, it is necessary to trace the development and use of software through the model. First, consider a software module, written conventionally. This conventional software will cost, as per the example, $25,000, resulting in a profit of $25,000 on the first, and only, use of the software. When a similar project, returning similar payoffs and requiring similar software, is assigned, an additional design effort is required, meaning another $25,000 expense to realize another $50,000 payoff. This means, that in two uses of the same or similar software, a profit of $50,000 is realized.

This is as if the software engineer took the path from node A to node B on every decision. The model indicates

that each time the engineer takes this path, he will realize a profit of $25,000.

Had the software engineers originally chosen to write a reusable module, the software would have cost $31,000, i.e., $30,000 to design and write the software and $1,000 for the fruitless library search. The resulting profit is $19,000. In this case, a second assignment calling for similar software can be answered by conducting a library search, retrieving the reusable software module and installing it. In the worse case, this will cost $1,000 for the search, $1000 for sorting and $20,000 (modification required) for installing it. If the payoff for branch M is only $40,000, the profit is $18,000. That is $37,000 profit in two calls for the software module.

If, however, since the module is being modified, it can be made to more nearly meet the specifications, it is reasonable to surmise that the payoff might be more than that for an unmodified module. If it can be made to meet specifications exactly, it will return a payoff of $50,000. This results in a profit of $28,000 for the second use of this module. When added to the profit of the first use of the module, a combined of $47,000 is realized, still below the profit of twice using the conventional modules.

However, on the third use of the modules, the conventional module will return another $25,000, but the reusable module is now coming into its own. Since it was

previously modified, the cost of using this module is only $7,000 ($1,000 for the search, $1,000 for sorting the retrieved modules and $5,000 for installing the reusable module). This returns $43,000 for the third use, alone and, when combined with the previous uses of the module, returns a gross profit of $90,000, compared to $75,000 profit for the conventional modules. The break-even point will occur when the gross profit exceeds the start-up costs of the library.

Even though these numbers are arbitrarily chosen, this discussion shows how the model can be used by a software engineering firm to predict its own break-even point in determining whether it should strive for reusability in software. It also shows how the model can provide the software engineering firm information to make the decision whether reuse is the right decision for the firm.

In his article in Computerworld [Ref. 14], Jones implies that the project initializing a reusable module will reap no benefit from writing reusable software and that only subsequent users of the software module will gain any benefit. The model supports Jones' assertions and gives an indication of the number of subsequent uses that must be realized for reusability to be of some benefit.

If the firm can predict that the software will not be reused a sufficient number of times to gain this advantage, it may determine that reusable software holds no advantage.

Further, there is the issue of the value of money. A return of $50,000, ten years hence is not the same as a return of $50,000, one year from now.

Here is a case in which a firm may not wish to pursue the reusable software course of action. The model has shown that it is possible to envision a circumstance in which reusability is, economically, a mistake. A software module that is only expected to be used once, or one that is not expected to be used within a reasonable frame of time, will provide no economic reuse advantage to the software firm producing it. This idea disputes the trend in the current literature implies that reusability is the remedy of today's software engineering dilemma.

VII. CONCLUSIONS

The issue of software reusability has received much attention in recent literature. The literature extols the virtues of reusable software, but fails to discuss analysis of the economic issues relevant to the area of software reusability. The model proposed in this thesis provides a methodology for explicitly studying these issues.

The model demonstrates the decision making process through which the software engineer progresses. It also provides guidance in how to establish incentives in order to establish a library of reusable modules and how long those incentives should be in place. The model demonstrates that, even though not initially profitable, after a point the reusable module can be an extremely profitable asset.

On the other hand, the model shows that reusability is not the panacea one is led to believe. As demonstrated, it is conceivable that some software firms may find it an economic disaster to pursue reusability if the software is not expected to be reused often enough or soon enough. The model also provides a means of determining how many times the software must be reused in order to be profitable.

The model analyzes the question of software reusability only from the standpoint of dollar value to the ogranization. This is not the only view to be taken. There may exist non-economic considerations that are not

incorporated into the model. For instance, an organization may consider that its programming expertise or its ability discover new and better solutions to old problems are critically important assets. The organization may therefore choose a conventional approach over the less costly avenue of reuse in order to maintain these important skills. Even though the model does not incorporate these considerations, it will allow the organization to determine a trade-off between economic and non-economic considerations.

The discussion of the model is meant to be suggestive, rather than inclusive; the figures used in this thesis were arbitrarily chosen to illustrate the utility of the model as an analytical tool. There is much more that can be done with this model. In the future, for example, thesis students may apply empirical data to the model to further explore its utility. Government agencies attempting to foster software reusability may use the model to ascertain why the agencies' software engineers are writing conventional software modules. Finally, private industry may use the model in actual cases to further verify the claims of this thesis.

APPENDIX

(Computer Programs)

```
Program DecisionTree;
var
(*
    Costs      Expected      Payoffs      Probabilities
                Values
 *)
     Cab,          Rb,           Vbr,           Pf,
     Cac,          Rc,           Vbnr,          Pm,
     Cbr, Cbnr,    Rd,           Ver,
     Cer, Cenr,    Re,           Venr,
     Cfm, Cfnm,    Rf,           Vem,
     Cgr, Cgnr,    Rg,           Venm,
     Cdf,                        Vfm,
                                 Vfnm,
                                 Vgr,
                                 Vgnr:                    real;

     outfile:                                             text;


function Max (a,b: real): real;
begin (* function Max *)
 if a >= b then
   Max := a
 else
   Max := b;
end (* function Max *);

begin (* main program *)

  assign (outfile,'dectre.dta'); rewrite (outfile);

  Cab := 0000.00; Cac := 1000.00; Cbr := 30000.00; Cbnr :=
25000.00;
  Cer := 30000.00; Cenr := 25000.00; Cfm := 20000.00; Cfnm
:= 5000.00;
  Cgr := 30000.00; Cgnr := 25000.00; Cdf := 1000.00;


  Vbr := 50000.00; Vbnr := 50000.00; Ver := 50000.00; Venr
:= 50000.00;
  Vfm := 40000.00; Vfnm := 40000.00; Vgr := 50000.00; Vgnr
:= 50000.00;

  writeln ('Pm':7,'Rb':11,'Rc':13,'Branch Selected':26);
  writeln (outfile,'Pm':7,'Rb':11,'Rc':13,'Branch
Selected':26);
```

```
  Pf := 0.75; Pm := 0.00;

  while Pm <= 1.00 do
    begin
    Rg := Max((Vgr - Cgr), (Vgnr - Cgnr));
    Re := Max((Ver - Cer), (Venr - Cenr));
    Rb := Max((Vbr - Cbr), (Vbnr - Cbnr));
    Rf := Pm * (Vfm - Cfm) + (1-Pm) * (Vfnm - Cfnm);
    Rd := Max(Rg, (Rf - Cdf));
    Rc := Pf * Rd + (1-Pf) * Re;

    write (Pm:8:2,Rb:13:2,(Rc-Cac):13:2);
    write (outfile,Pm:8:2,Rb:13:2,(Rc-Cac):13:2);

    if Rb >= (Rc - Cac) then
      begin
        writeln ('NS':16);
        writeln (outfile,'NS':16);
      end
    else
      begin
        writeln ('S':16);
        writeln (outfile,'S':16);
      end;

    Pm := Pm + 0.01;

    end; (* while *)

  close (outfile);
end.
```

Pm	Rb	Rc	Branch Selected
0.00	25000.00	30750.00	S
0.01	25000.00	30637.50	S
0.02	25000.00	30525.00	S
0.03	25000.00	30412.50	S
0.04	25000.00	30300.00	S
0.05	25000.00	30187.50	S
0.06	25000.00	30075.00	S
0.07	25000.00	29962.50	S
0.08	25000.00	29850.00	S
0.09	25000.00	29737.50	S
0.10	25000.00	29625.00	S
0.11	25000.00	29512.50	S
0.12	25000.00	29400.00	S
0.13	25000.00	29287.50	S
0.14	25000.00	29175.00	S
0.15	25000.00	29062.50	S
0.16	25000.00	28950.00	S
0.17	25000.00	28837.50	S
0.18	25000.00	28725.00	S
0.19	25000.00	28612.50	S
0.20	25000.00	28500.00	S
0.21	25000.00	28387.50	S
0.22	25000.00	28275.00	S
0.23	25000.00	28162.50	S
0.24	25000.00	28050.00	S
0.25	25000.00	27937.50	S
0.26	25000.00	27825.00	S
0.27	25000.00	27712.50	S
0.28	25000.00	27600.00	S
0.29	25000.00	27487.50	S
0.30	25000.00	27375.00	S
0.31	25000.00	27262.50	S
0.32	25000.00	27150.00	S
0.33	25000.00	27037.50	S
0.34	25000.00	26925.00	S
0.35	25000.00	26812.50	S
0.36	25000.00	26700.00	S
0.37	25000.00	26587.50	S
0.38	25000.00	26475.00	S
0.39	25000.00	26362.50	S
0.40	25000.00	26250.00	S
0.41	25000.00	26137.50	S
0.42	25000.00	26025.00	S
0.43	25000.00	25912.50	S
0.44	25000.00	25800.00	S
0.45	25000.00	25687.50	S
0.46	25000.00	25575.00	S
0.47	25000.00	25462.50	S
0.48	25000.00	25350.00	S
0.49	25000.00	25237.50	S
0.50	25000.00	25125.00	S

0.51	25000.00	25012.50	S
0.52	25000.00	24900.00	NS
0.53	25000.00	24787.50	NS
0.54	25000.00	24675.00	NS
0.55	25000.00	24562.50	NS
0.56	25000.00	24450.00	NS
0.57	25000.00	24337.50	NS
0.58	25000.00	24225.00	NS
0.59	25000.00	24112.50	NS
0.60	25000.00	24000.00	NS
0.61	25000.00	24000.00	NS
0.62	25000.00	24000.00	NS
0.63	25000.00	24000.00	NS
0.64	25000.00	24000.00	NS
0.65	25000.00	24000.00	NS
0.66	25000.00	24000.00	NS
0.67	25000.00	24000.00	NS
0.68	25000.00	24000.00	NS
0.69	25000.00	24000.00	NS
0.70	25000.00	24000.00	NS
0.71	25000.00	24000.00	NS
0.72	25000.00	24000.00	NS
0.73	25000.00	24000.00	NS
0.74	25000.00	24000.00	NS
0.75	25000.00	24000.00	NS
0.76	25000.00	24000.00	NS
0.77	25000.00	24000.00	NS
0.78	25000.00	24000.00	NS
0.79	25000.00	24000.00	NS
0.80	25000.00	24000.00	NS
0.81	25000.00	24000.00	NS
0.82	25000.00	24000.00	NS
0.83	25000.00	24000.00	NS
0.84	25000.00	24000.00	NS
0.85	25000.00	24000.00	NS
0.86	25000.00	24000.00	NS
0.87	25000.00	24000.00	NS
0.88	25000.00	24000.00	NS
0.89	25000.00	24000.00	NS
0.90	25000.00	24000.00	NS
0.91	25000.00	24000.00	NS
0.92	25000.00	24000.00	NS
0.93	25000.00	24000.00	NS
0.94	25000.00	24000.00	NS
0.95	25000.00	24000.00	NS
0.96	25000.00	24000.00	NS
0.97	25000.00	24000.00	NS
0.98	25000.00	24000.00	NS
0.99	25000.00	24000.00	NS
1.00	25000.00	24000.00	NS

```
Program DecisionTree1;

type
  pathtype = (S, NS);

var
(*
    Costs      Expected      Payoffs      Probabilities
               Values
 *)
     Cab,         Rb,           Vbr,           Pf,
     Cac,         Rc,           Vbnr,          Pm,
     Cbr, Cbnr,   Rd,           Ver,
     Cer, Cenr,   Re,           Venr,
     Cfm, Cfnm,   Rf,           Vem,
     Cgr, Cgnr,   Rg,           Venm,
     Cdf,                       Vfm,
                                Vfnm,
                                Vgr,
                                Vgnr:                    real;


     outfile:                                            text;


     path,
     lastpath:
pathtype;

function Max (a,b: real): real;
begin (* function Max *)
 if a >= b then
   Max := a
 else
   Max := b;
end (* function Max *);

procedure Output(parameter1, parameter2, parameter3:real);
begin (* procedure Output *)
  writeln (parameter1:8:2, parameter2:8:2,parameter3:15:2);
  writeln (outfile,parameter1:8:2,
parameter2:8:2,parameter3:15:2);
end; (* procedure Output *)

begin (* main program *)

  assign (outfile,'thesis.dta'); rewrite (outfile);

  Cab := 0000.00; Cac := 1000.00; Cbr := 30000.00; Cbnr :=
25000.00;
  Cer := 30000.00; Cenr := 25000.00; Cfm := 20000.00; Cfnm
:= 5000.00;
```

```
  Cgr := 30000.00; Cgnr := 25000.00; Cdf := 1000.00;


  Vbr := 50000.00; Vbnr := 50000.00; Ver := 50000.00; Venr
:= 50000.00;
  Vfm := 40000.00; Vfnm := 40000.00; Vgr := 50000.00; Vgnr
:= 50000.00;


  Pm := 0.00;

  writeln ('Pm':7,'Pf':8,'Expected Value at A':22);
  writeln (outfile,'Pm':7,'Pf':8,'Return':15);

  while Pm <= 1.00 do
    begin
    Pf := 0.00;
    lastpath := NS;

  while Pf <= 1.00 do
    begin
    Rg := Max((Vgr - Cgr), (Vgnr - Cgnr));
    Re := Max((Ver - Cer), (Venr - Cenr));
    Rb := Max((Vbr - Cbr), (Vbnr - Cbnr));
    Rf := Pm * (Vfm - Cfm) + (1-Pm) * (Vfnm - Cfnm);
    Rd := Max(Rg, (Rf - Cdf));
    Rc := Pf * Rd + (1-Pf) * Re;

    if Rb >= (Rc - Cac) then
      path := NS
    else
      path := S;

    if (path = S) and (lastpath = NS) then
      Output(Pm,Pf,Rc-Cac);

    lastpath := path;
    Pf := Pf + 0.01;

    end; (* while *)

    Pm := Pm + 0.01;

    end; (* while *)

  close (outfile);
end.
```

Pm	Pf	Return
0.00	0.12	25080.00
0.01	0.12	25062.00
0.02	0.12	25044.00
0.03	0.12	25026.00
0.04	0.12	25008.00
0.05	0.13	25072.50
0.06	0.13	25053.00
0.07	0.13	25033.50
0.08	0.13	25014.00
0.09	0.14	25071.00
0.10	0.14	25050.00
0.11	0.14	25029.00
0.12	0.14	25008.00
0.13	0.15	25057.50
0.14	0.15	25035.00
0.15	0.15	25012.50
0.16	0.16	25056.00
0.17	0.16	25032.00
0.18	0.16	25008.00
0.19	0.17	25045.50
0.20	0.17	25020.00
0.21	0.18	25053.00
0.22	0.18	25026.00
0.23	0.19	25054.50
0.24	0.19	25026.00
0.25	0.20	25050.00
0.26	0.20	25020.00
0.27	0.21	25039.50
0.28	0.21	25008.00
0.29	0.22	25023.00
0.30	0.23	25035.00
0.31	0.23	25000.50
0.32	0.24	25008.00
0.33	0.25	25012.50
0.34	0.26	25014.00
0.35	0.27	25012.50
0.36	0.28	25008.00
0.37	0.29	25000.50
0.38	0.31	25023.00
0.39	0.32	25008.00
0.40	0.34	25020.00
0.41	0.36	25026.00
0.42	0.38	25026.00
0.43	0.40	25020.00
0.44	0.42	25008.00
0.45	0.45	25012.50
0.46	0.48	25008.00
0.47	0.52	25014.00
0.48	0.56	25008.00
0.49	0.61	25006.50
0.50	0.67	25005.00

0.51	0.75	25012.50
0.52	0.84	25008.00
0.53	0.96	25008.00

LIST OF REFERENCES

1. Johnson, LCDR William C., Reusable Software, Master's Thesis, Naval Postgraduate School, Monterey, California, p. 7, March 1984.

2. Jones, T. C., "Laying the Groundwork With Reusable Code", Computerworld, v. 18, no. 26Am, p. 12, 27 June 1984.

3. Anonymous, "Softalk: Keeping Maintenance Minimal", Computerworld, v. 19, no. 5, p. 37, 4 February 1985.

4. Romberg, F.A. and Thomas, A.B.; "Reusable Code, Reliable Software," Computerworld, Vol. 18, No. 13, p. 22ID, 26 March 1984.

5. Ramamoorthy, C. V., Prakash, A., Tsai, W. T., Usada, Y., "Software Engineering: Problems and Perspectives", Computer, v. 17, no. 10, pp. 191-209, October 1984.

6. Romberg, F. A. and Thomas, A. B.; "Reusable Code, Reliable Software", Computerworld, v. 18, no. 13, p. 22ID, 26 March 1984.

7. King, David, Current Practices in Software Development, p. 175, Yourdon Press, 1984.

8. Heffernan, Henry, "Modular Software Simplifies Maintenance and Reuse", Government Computer News, v. 5, no. 21, p.70, 24 October 1986.

9. Ackoff, Russell, L. and Sasieni, Maurice W., Fundamentals of Operations Research, John Wiley & Sons, Inc., 1968

10. Markland, Robert E. and Sweigart, James R., Quantitative Methods: Applications to Managerial Decision Making, John Wiley & Sons, Inc., 1987

11. Boehm, Barry W., The Hardware/Software Cost Ratio: Is It a Myth?, letter to the editor of Computer, March 1983, p. 79.

12. Jones, T.C., "Laying the Groundwork With Reusable Code," Computerworld, Vol. 18, No. 26Am, June 27, 1984, p. 13.

13. Jones, T. C., "Laying the Groundwork With Reusable Code", Computerworld, v. 18, no. 26Am, p. 13, 27 June 1984.

14. Jones, T. C., "Laying the Groundwork With Reusable Code", Computerworld, v.18, no. 26Am, p. 13, 27 June 1984.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Vincent Y. Lum, Chairman Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	1
4.	Prof. Gordon Bradley, Code 52Bz Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	2
5.	Joel Trimble STARS Program Office of Secretary of Defense (R & AP/CET) Room 3D139 (1211 FERN, C112) Pentagon Washington, D. C. 20301	1
6.	Dr. E. Royce, Code 38 Naval Weapons Center China Lake, California 93950	1
7.	Daniel L. Davis MBARI 160 Central Avenue Pacific Grove, California 93950	1
8.	Dr. Gregory Aharonian Source Translation and Optimization P. O. Box 404 Belmont, Massachusetts 02178	1
9.	LCDR William D. Randall, Jr. Engineering and Weapons Division United States Naval Academy Annapolis, Maryland 21402	3